UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology

Carl Hjalmar Love Hult

Using and Evaluating the Real-time Spatial Perception System Hydra in Real-world Scenarios

Bachelor's Thesis (12 ECTS)

Curriculum Science and Technology

Supervisor:
Associate professor of robotics engineering Karl Kruusamäe, PhD

Tartu 2023

Abstract

Using and Evaluating the Real-time Spatial Perception System Hydra in Real-world Scenarios

Hydra is a real-time machine perception system released open source in 2022 as a package for Robot Operating System (ROS). Machine perception systems like Hydra may play a role in the engineering of the next generation of spatial AIs for autonomous robots. Hydra is in the preliminary stages of its existence and does not come with intrinsic support for running on custom datasets. This thesis primarily aims to find out whether the promised capabilities of Hydra can be replicated. As well as to establish a workflow and guidelines for what modifications to Hydra are needed to successfully run it.

Keywords: Hydra, Machine perception, Spatial perception, Mapping, Spatial AI

CERCS: T111 - Imaging, image processing, T120 - Systems engineering, computer technology, T125 - Automation, robotics, control engineering

Kokkuvõte

Reaalajas toimiva ruumilise taju süsteemi Hydra kasutamine ja hindamine praktilistes stsenaariumides

Hydra on masintaju süsteem, mis avaldati aastal 2022 ning on kättesaadav ROS-i paketina avatud lähtekoodi formaadis. Hydra sarnased süsteemid omavad potentsiaalset rolli järgmise põlvkonna ruumilise tehisintellekti ja autonoomsete robotite väljatöötamisel. Süsteem on alles algstaadiumis ning ei toeta kohandatud andmekogumite kasutamist. Käesoleva bakalaureusetöö peamine eesmärk on uurida, kas Hydra süsteemi lubatud omadusi on võimalik samaväärselt jäljendada ning luua suunised ja töövoog Hydra modifitseerimiseks ja selle edukaks kasutamiseks.

Võtmesõnad: Hydra, masintaju, ruumiline taju, kaardistamine, ruumiline tehisintellekt

CERCS: T111 Pilditehnika, T120 - Süsteemitehnoloogia, arvutitehnoloogia, T125-Automatiseerimine, robootika, juhtimistehnika

Table of Contents

Introduction

The movie Terminator directed by James Cameron in 1984 starring Arnold Schwarzenegger portrays a cybernetic robot: the Terminator, sent back in time to assassinate Sarah Connor to prevent the artificial intelligence (AI) overlord Skynet from being overthrown in the future. While the Terminator acts as an antagonist in the movie, he is amongst other things, a robot capable of performing human-like if not superior to human-like actions in the real world like riding motorcycles and navigating complex environments.

AI has recently become part of the global zeitgeist due to the advent of highly advanced chat bots like ChatGPT [1]. While AI has rapidly advanced on the intellectual level, the same cannot be said for the AIs we need to build autonomous robots, and hopefully the nice kinds of robots in this case. The problem boils down to the fact that machines fundamentally lack rationale. And it is indeed difficult to break down even the most simple processes to a non-rational entity like a robot [2].

A part of the solution needed is advanced machine-perception systems as they will lay the foundations for future autonomous robotic AIs. At the basic level, good sensors, computers and some system to make sense of all the sensory input is needed. One proposed system is Hydra, a spatial perception system developed at Massachusetts Institute of Technology, partially funded by the United States Air Force. Hydra was recently released in an open-source format as a collection of packages for Robot Operating System. However, Hydra was not released completely open source. Hence, the goal of this thesis is to find out whether the demonstrated capabilities of Hydra can be replicated with limited access to all of Hydra.

1. Background

For a long time, robots have been limited to applications in highly predictable and repeatable industrial processes. The first robots were factory assembly line robots [3]. But as technology and knowledge have advanced, so have the potential use-cases for robots. No longer are robots strictly limited to applications in assembly lines, but have begun to branch into education, food delivery, remote aid delivery and even farming with the advent of autonomous mobile robots [4].

Autonomous robots need to be able to perceive and understand their environment to work in them. Hydra is a machine perception system that is able answer the questions of *where* a robot is, as well as *what* and *who* is around the robot [5]. Hydra does this by building a hierarchical digital representation of the real world in the form of a 3D Scene Graph, separating buildings, rooms, objects, people, and places. The rest of this introductory segment is dedicated to reviewing the core problems, and some of the technologies used either as part of Hydra or as part of machine perception in general. First, that of localization, then mapping, a brief introduction to simultaneous localization and mapping, and finally background on 3D Scene Graphs and a brief technical review of Hydra.

1.1. Localization

Arguably, one of the most fundamental aspects of machine perception is answering the question of where the robot is [6]. This is the problem of localization.

1.1.1. Satellite navigation

Possibly the most universally applied localization technique is satellite navigation. Finding use in everything from smartphones to cars and robotic lawn mowers [7]. Satellite navigation solutions commonly have the advantage of working globally and with a somewhat good degree of accuracy, GPS-enabled smartphones typically have an accuracy radius within a 4.9 m [8]. That degree of accuracy may however be insufficient for localization applications indoors, because rooms are typically on the scale of 5 m.

1.1.2. Beacon-based localization

By placing beacons in a space with known positions, it is possible to locate a robot through its relation to the beacons [9]. This approach can be accurate [10], but might require an unobstructed line of sight between the beacons and the robot depending on the signal used to identify the beacons or robot, a camera that works in the visible light spectrum cannot see through walls. A disadvantage to beacon-based localization is the requirement for beacons which need to be installed prior to the robot [11]. This also implies that the approach is unviable in unknown environments.

1.1.3. Odometry or Dead reckoning

If a mobile robot has wheels, then estimating its location can be done easily by counting how many times its wheels have spun [6]. It if has legs then the number of steps taken can be counted etc. While this approach is seemingly simple and might not require external equipment, the fact that even miniscule inaccuracies in sensor precision eventually lead to large errors over time makes this approach problematic for long term localization.

1.1.3.1. Inertia

A sensor commonly used in robotics is the Inertial Measurement Unit - IMU. With an IMU it is possible to measure the acceleration of the robot along with its orientation. This can also be considered odometry [6]. Integrating the acceleration results in a velocity estimate, which can also be integrated to estimate the pose of the robot. While this approach is simple to integrate

and is able to detect very small changes in acceleration in short time intervals, it suffers greatly from inaccuracy over time [12].

1.1.4. Visual Odometry

Visual Odometry is an approach to localization that makes an estimate of the robot's odometry based on visual information. This is achieved by analyzing the latest sequence of images that the robot's camera captures, noting important landmarks in those pictures and deducing where the robot moved based on where those landmarks have moved in relation to the robot. This approach is less susceptible to error accumulation over time than the two former methods, but will still suffer from inaccuracy over time [13], [14].

1.1.5. Kimera Visual Inertial Odometry

The combination of high resolution data from the IMU with the accurate odometry estimate from the visual part results in a very accurate pose estimate [14]. But again, small errors eventually lead to great inaccuracies. Released as part of the Open-Source library for Real-Time Metric-Semantic Localization and Mapping by Rosinol et al., Kimera Visual Inertial Odometry - Kimera-VIO is a GTSAM-based approach to Visual Inertial Odometry and was shown to have a drift of less than 4 cm in a 32 m trajectory [15]. It was used for estimating an odometric trajectory in the Hydra paper [5]. And it is also used in this thesis for odometric estimation.

1.2. Mapping

Mapping is the process of making a digital representation of an environment. To make a map, the distance from the robot needs to be measured to features around the robot, like walls, doors, chairs etc. This can be done with any sensor capable of measuring distance like lidar, ultrasonic sensors or dual cameras. There are many different types of maps in use in robotics today [16]–[19]. A few of them along with their uses are presented here in this chapter.

1.2.1. Metric maps

A metric map, or geometric map is a map that displays the features of things in terms of geometry and their relation to each other geometrically. One way to make a metric map is by constructing an occupancy grid. Essentially, an occupancy grid is a representation of an environment simply in terms of whether a space is occupied by something or not. By recording occupied and non-occupied spaces a map similar to that of which is illustrated in figure 1 can be achieved. These maps are very useful when it comes to pathfinding as clear paths vs non-clear paths are clearly distinguished [14].

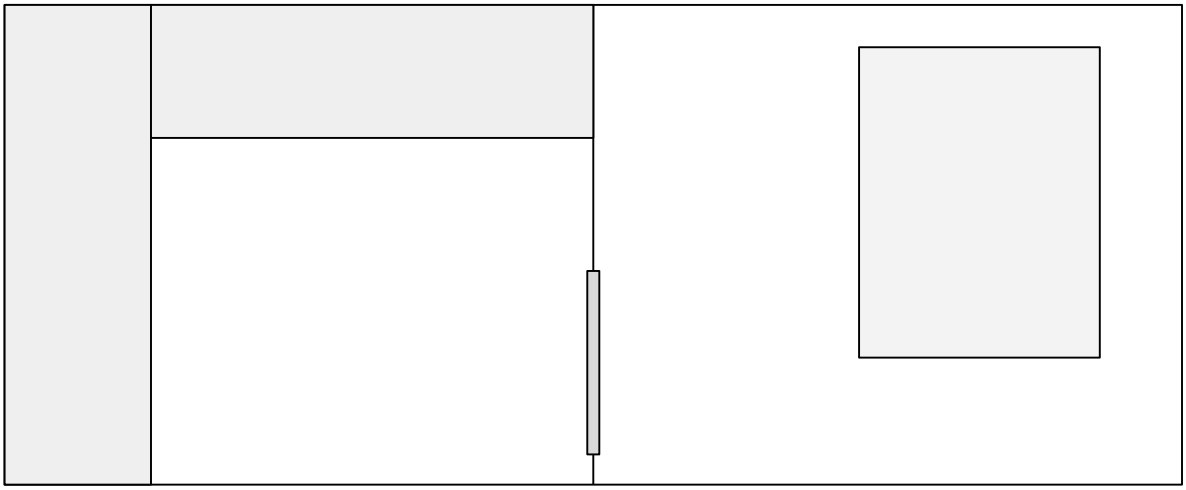

Figure 1. *A 2-dimensional metric occupancy grid map*

1.2.2. Topological maps

Topological maps are maps concerned with the relationship between landmarks and how they are connected [14], [20]. In topological maps, landmarks are usually represented as points of interest and are connected via arcs; descriptions of how to navigate from one point to another [20]. Topological maps are especially useful for high-level path planning [14], as points are described with actual names. For example:

Path planning in a strictly metric map: *go from x:102, y:332 to x:203, y:111*.
Path planning in a strictly topological map: *go from the sink to the bed*.

Illustrated in Figure 2 is a strictly topological map.

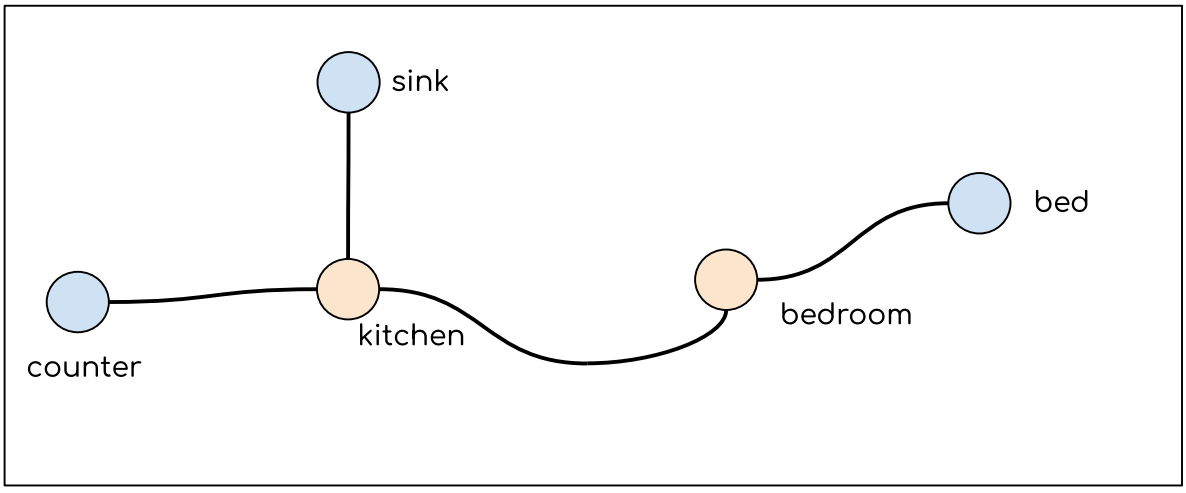

Figure 2. *A 2-dimensional topological map*

1.2.3. Topometric maps

A topometric map is a map which combines a metric map and a topological map [21]. The advantage of a topometric map is that it contains both the accurate description of a metric map, and the high-level abstract relationships between places [14]. Topometric maps have been used to improve path finding [22]. A topometric map is illustrated in Figure 3.

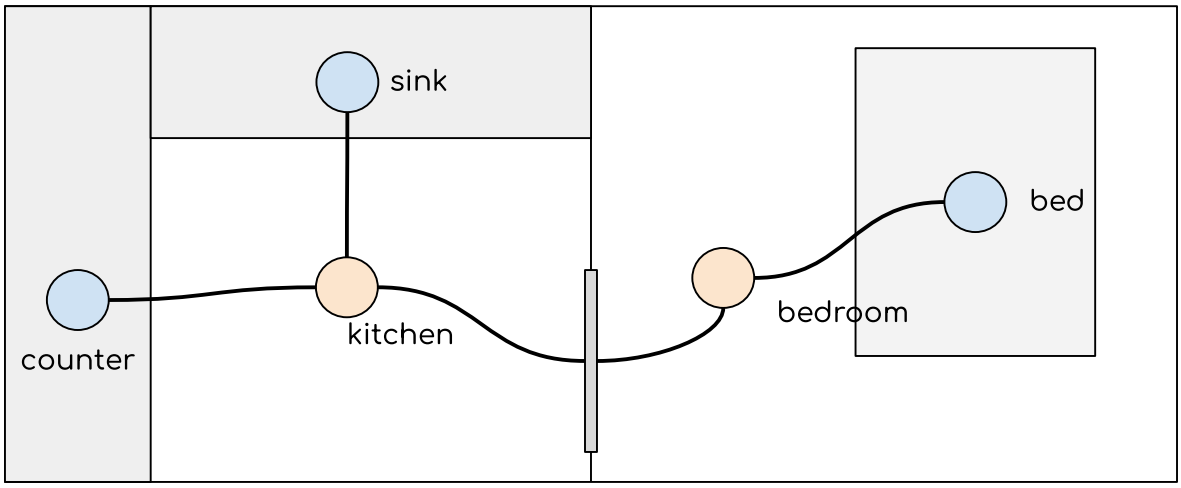

Figure 3. *A 2-dimensional topometric hybrid map*

1.2.3. Metric-semantic maps

Thus far, several solutions have been proposed to address the question of answering *where* the surroundings are. But what if the robot also had information about what was around it? Incorporating the semantics about objects and entities would enable new types of decisions. While a simple geometric map might equate a closed door to a wall, a representation that incorporates semantics and geometry, might contain the additional information that the door the robot is seeing is an actionable object that can be opened [23]. Figure 4 presents a metric-semantic map.

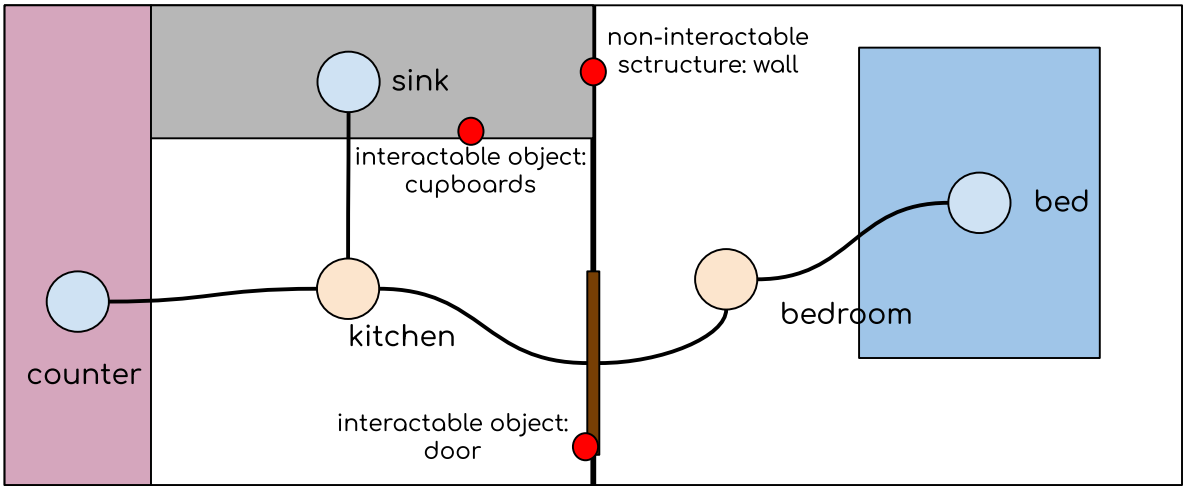

Figure 4. *A metric-semantic map with topology*

1.2.4. Simultaneous localization and mapping - SLAM

But how do we build the maps? Simultaneous Localization and Mapping - SLAM. is a term coined in 1995 by Durrant-Whyte et al. In essence, SLAM allows a mobile robot after being placed in a foreign environment, to gradually map it while accounting for its own position in the map it builds [24]. The development of new algorithms capable of solving the SLAM problem is ongoing at pace with the development of new and better sensors [25], but can largely be considered a solved problem [24]. SLAM is used extensively today, with applications ranging from robotic vacuum cleaners to autonomous package delivery drones [26]. Still, as explained in the section 1.2 on localization, no localization method is perfect and compounding error in position estimate will lead to compounding error in the map. This is illustrated in Figure 5 where a robot has a small sensor error leading to it thinking that it is drifting to the right, leading to a skewed map.

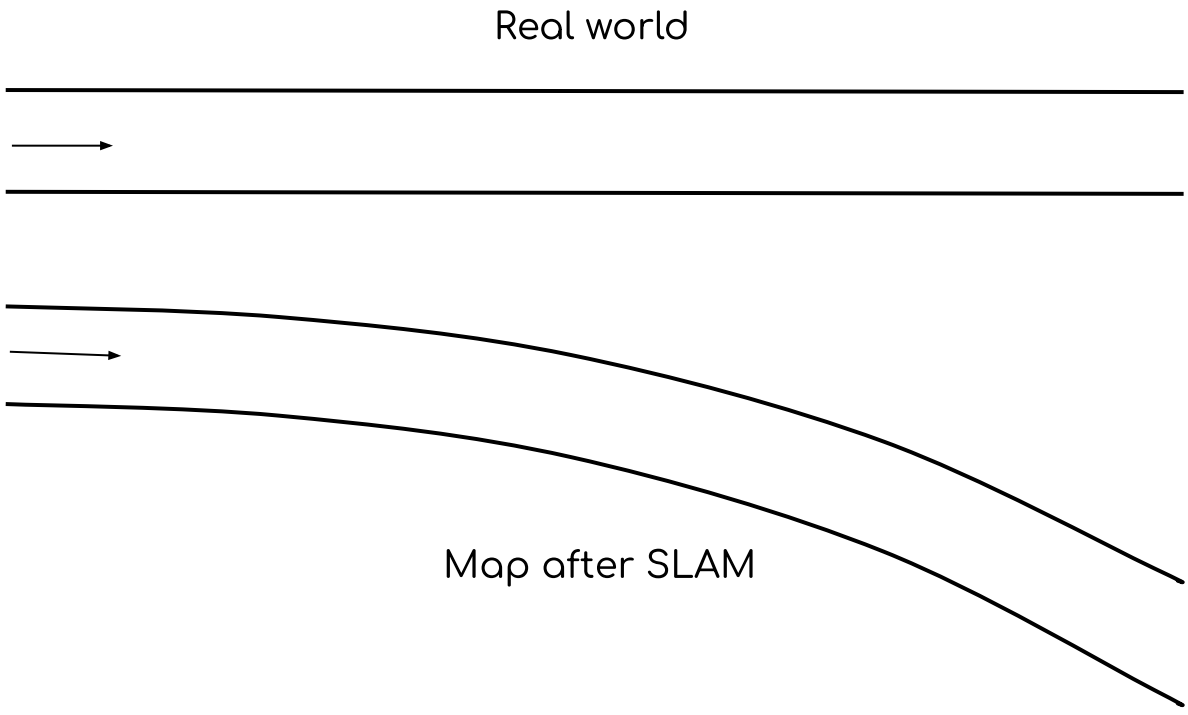

Figure 5. *The compounding error problem in SLAM.*

1.2.5. Loop closures

One way to account for the compounding error is by doing loop closures. The main idea of loop closures is to compare what I am seeing now, vs what I have seen before, and if those two match up to a certain degree - assume I am at the location I was before [14]. Figure 6 illustrates the concept of loop closures and map optimization. Doing loop closures by keeping a record of every recorded visited frame is naturally going to lead to a database of frames growing proportionally which might cause issues with growing databases, although implementations of loop closure detection have been made possible over 1000 km trajectories [27].

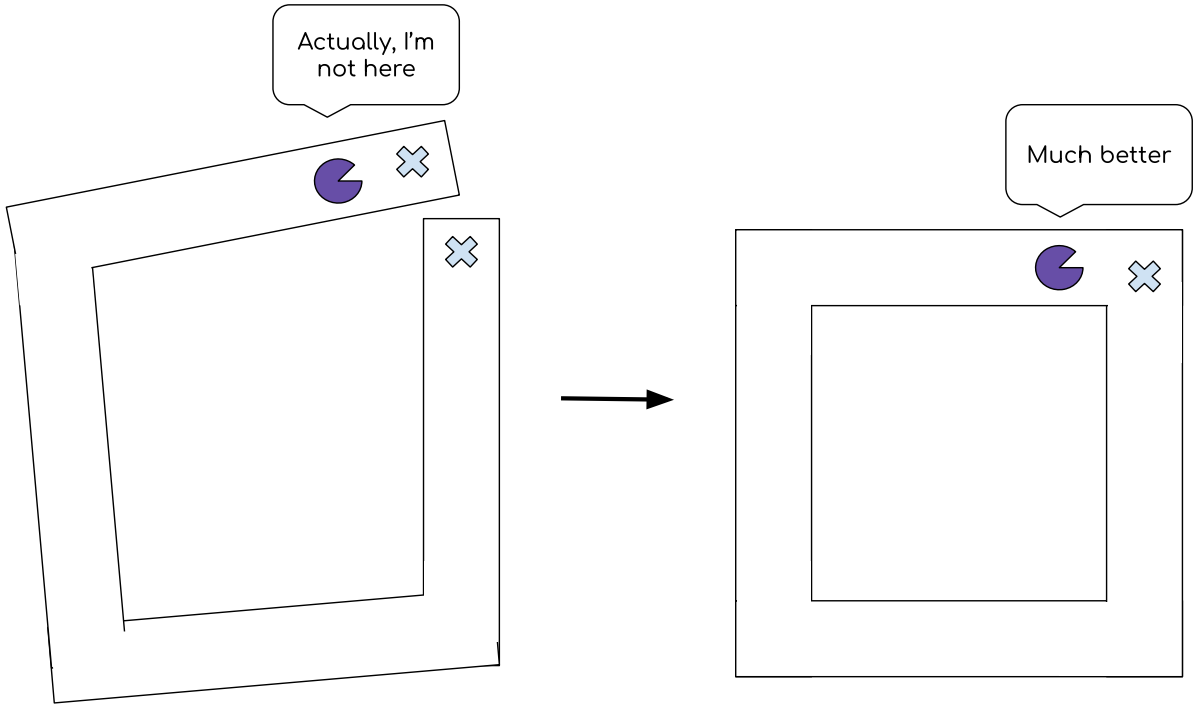

Figure 6. *Drift of estimated pose and the purpose of loop closure detection and map optimization*

1.2.6. 3D Dynamic Scene Graphs

A 3D Dynamic Scene Graph - DSG is a hierarchical representation of an environment.. Hierarchical in this sense means that it is a layered map with the different layers of abstraction representing different concepts. In the case of the following example – buildings, rooms, places and structures, objects and agents as topological maps, and a metric-semantic mesh. Layer 1 contains information on where the environment is. Layer 2 contains information on what is in the environment. Layers 3-5 contain navigational information on the environment - how empty spaces and rooms are connected. The Kimera DSG is featured in figure 7 [28], [29].

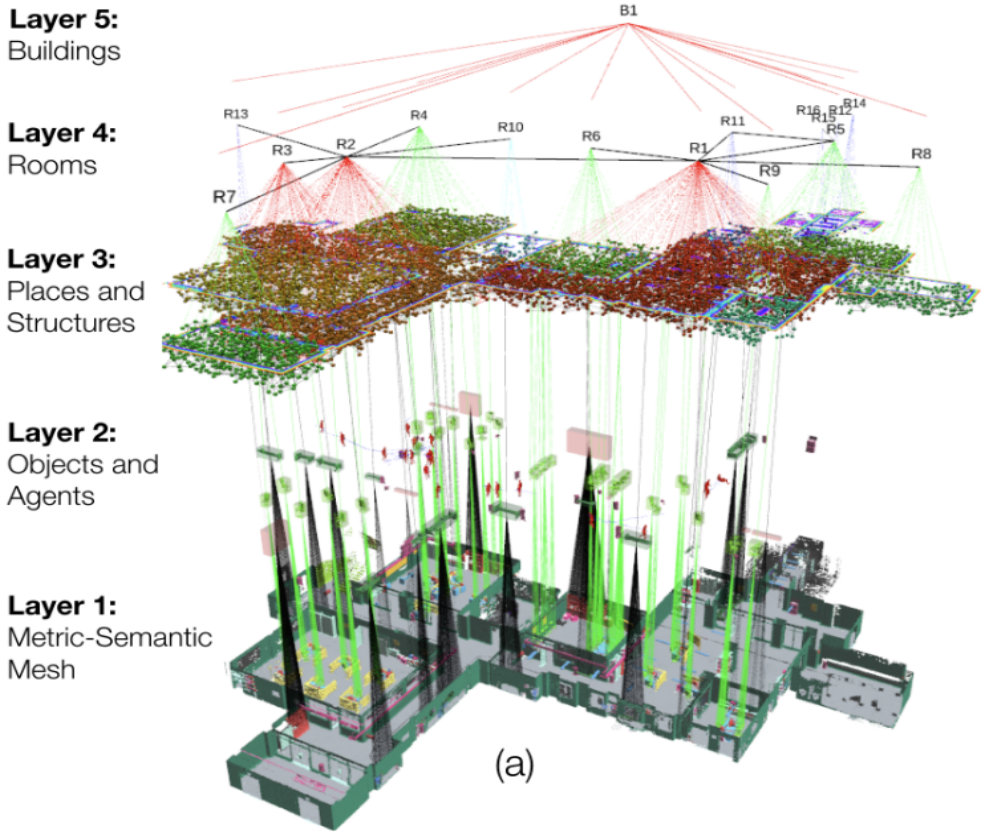

Figure 7. *Kimera's 3D Scene Graph [28].*

1.3. Overview of Hydra

Hydra is a real-time system for layering Buildings, Rooms, Places, Objects and Agents, and a Metric-Semantic 3D Mesh into a Dynamic 3D Scene Graph [5]. Hydra achieves this through the implementation of new real-time scene graph-construction algorithms and a new approach to loop closure detection in 3D Scene Graphs. An image of Hydra's scene graph is presented in figure 8.

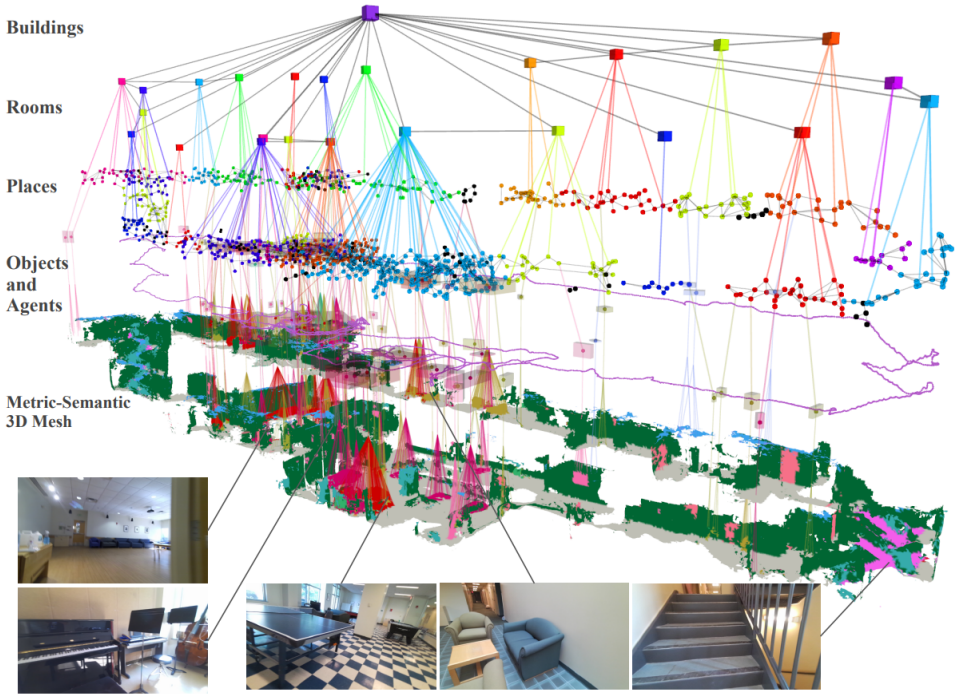

Figure 8. *Input data and Hydra's 3D scene graph* [5]

- Layer 1 - the metric-semantic mesh - is constructed by integrating a semantic point cloud generated from combining depth and semantic image with a distance field, then mesh reconstruction is done based on the result.
- Layer 2 - objects and agents - is constructed by segmenting objects based on the mesh.
- Layer 3 - places - is built by partitioning the mesh during construction and assigning nodes to places at a certain distance from obstacles.
- Layer 4 - rooms - is constructed based on layer 3 by incrementally pruning nodes from layer 3 and counting the number of connections until certain nodes can be considered to be part of a "community" of nodes.
- Layer 5 - buildings - is a node connected to every room [5].

Hydra performs loop closures by storing keyframes containing low to high-level descriptors in the agent layer. With every new keyframe, the appearance information is compared to the library. If a valid loop closure is detected, the scene graph is optimized and assembles into a coherent scene graph [5].

2. Aims and motivations of the thesis

Hydra is new software and its source code has only recently been made mostly publicly available. As of now, the only public examples of Hydra running are from the authors themselves. The official instructions and documentation on Hydra are related to running it on the uHumans2 dataset [1]. The uHumans2 dataset contains lots of ground-truth data and does not say alot about running Hydra on physical hardware.

Hence, the aims of this thesis are the following:

1. Establish a workflow on what is needed to successfully run Hydra.
2. Replicate the Hydra demo on a custom dataset.

[1] http://web.mit.edu/sparklab/datasets/uHumans2/

3. Experimental part

3.1. Robot Operating System

Hydra is available as a collection of packages for Robot Operating System (ROS). The general structure of ROS involves the creation of ROS-nodes that communicate with each other using specific messages over specific topics and handshake with the Master ROS node. Therefore, a ROS-node can act as a message-receiver (subscriber), a message-sender (publisher) or both. One of the advantages of this structure is that individual packages can be combined, and functionalities of existing packages can be extended by establishing communications between them and the other packages through the subscriber-publisher structure. This concept is further illustrated in figure 9.

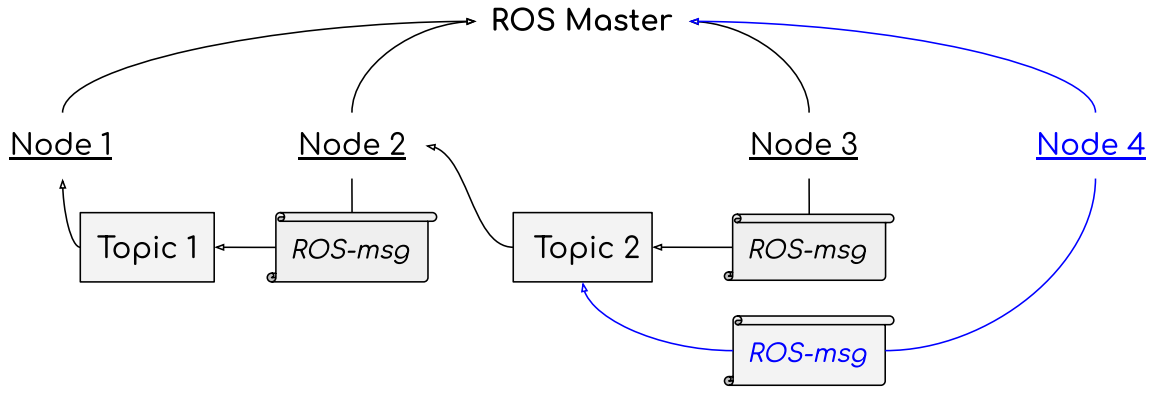

Figure 9. *The ROS structure. Node 1 is acting as a subscriber, Node 2 is acting as a subscriber and publisher, Node 3 is acting as a publisher. Node 4 serves the purpose of illustrating how new applications/nodes/data can be added and incorporated into an existing system.*

What this means practically and more relevantly in the case of this thesis is as long as there are publicly available packages, or a willingness to develop the packages that publish the right kind of messages. Establishing data feeds from sources that are not inherently supported should not require much or any actual editing of the code-base of Hydra.

What this thesis initially seeked to accomplish was to establish exactly how a pipeline for running Hydra successfully looks like. To provide further background on this topic, some flowcharts are presented in the next section.

3.1.1. Hydra data feed

In the case of running Hydra with the uHumans2 dataset without Kimera-VIO, the three essential data feeds being a Color image (can be either semantically segmented or just an unprocessed RGB image), Depth image, and Odometry are directly provided by the dataset. A flow chart of the data feed paths is provided in figure 10.

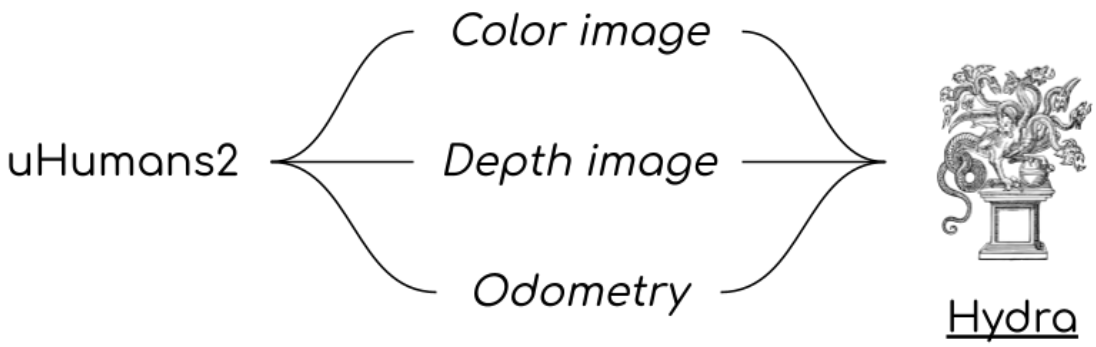

Figure 10. *Data feed from uHumans2 to Hydra. Data/ROS-messages are portrayed in cursive and ROS applications are underlined.*

It is also possible to run Hydra on the uHumans2 dataset with Kimera-VIO as uHumans2 also contains stereo camera and IMU data. In figure 11 the data feed is illustrated again but the new additions are marked in red:

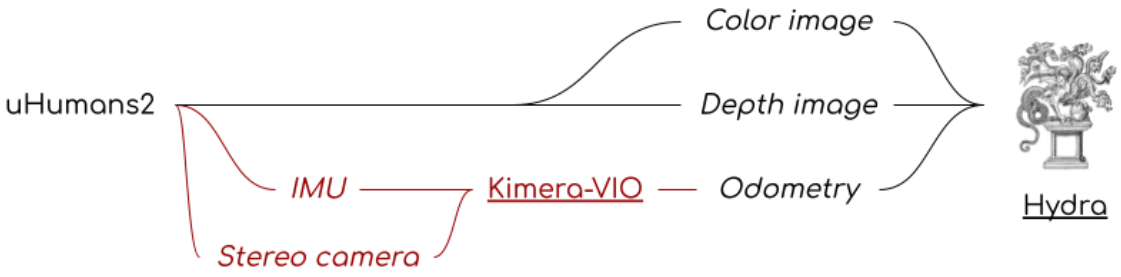

Figure 11. *Data feed from uHumans2 to Hydra using Kimera-VIO. Data/ROS-messages are portrayed in cursive and ROS applications are underlined.*

The complete established pipeline is presented in figure 12 in terms of data flow.

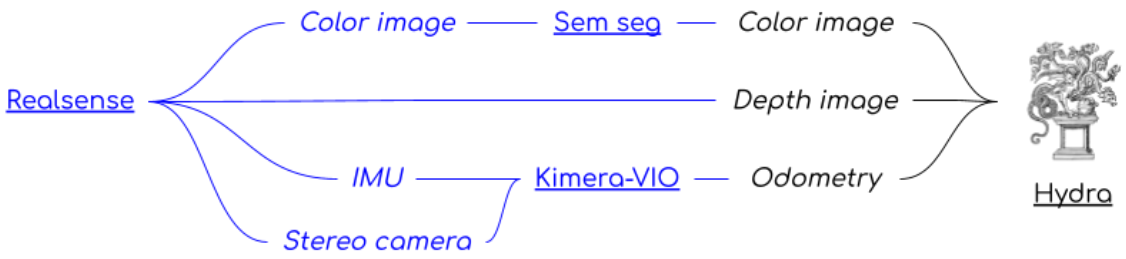

Figure 12. *Data feed from uHumans2 to Hydra using Kimera-VIO. Data/ROS-messages are portrayed in cursive and ROS applications are underlined. Sem seg is short for Semantic Segmentation.*

A more technical and detailed overview of the dataflow and the finished pipeline including real names of nodes and is provided in figure 13.

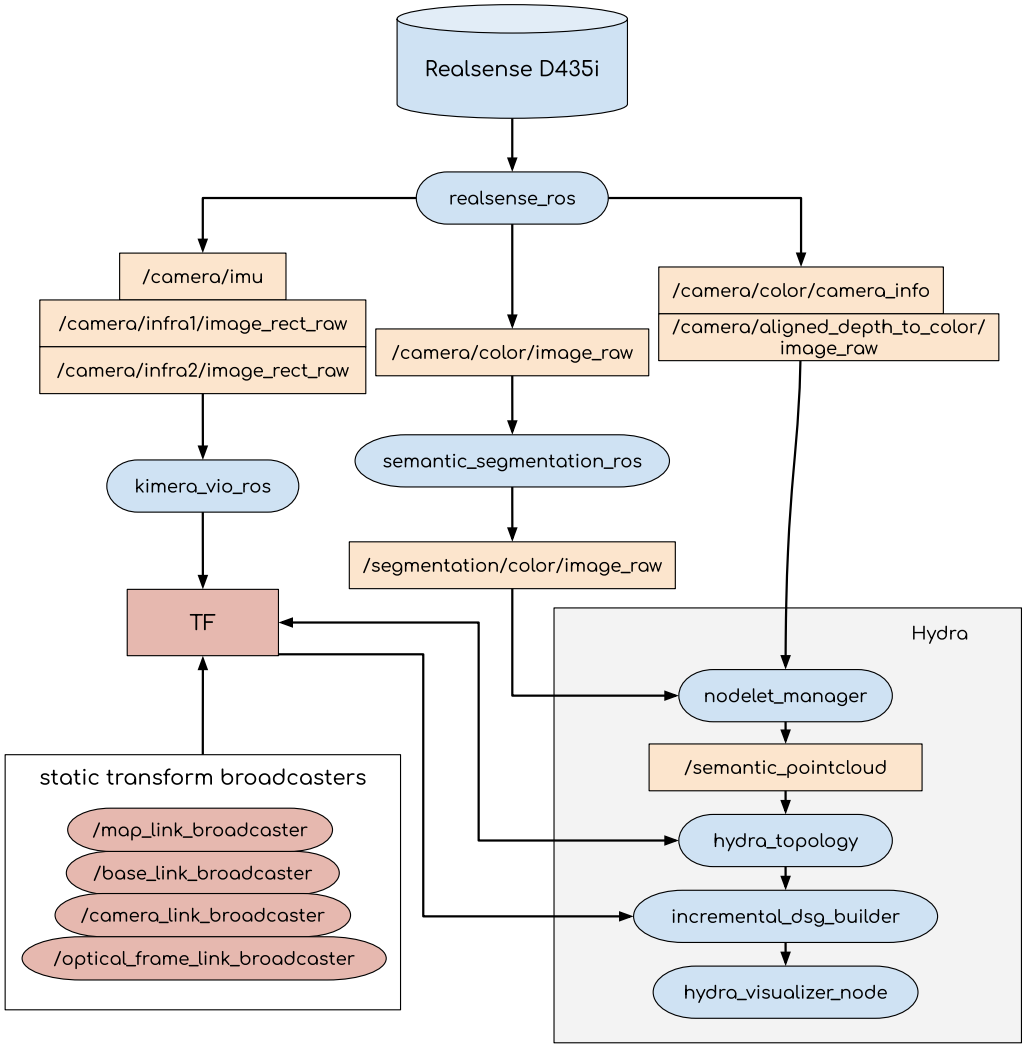

Figure 13. *An overview of the completed pipeline*

3.2. Basic setup requirements

All testing and direct work with Hydra was done on a workstation with an Intel Core i7 8700k, one Nvidia GTX3080, and 32 GB of ram natively running Ubuntu 20.04 Focal.

The sensory collection machine was a Lenovo ideapad 510S with an Intel Core i3-6100U cpu and 8 GB of ram.

The sensor used was Intel Realsense D435i. As for the choice of sensory unit, the reason is twofold.

Firstly, the Realsense D435i offers RGB, dual infrared-spectra cameras, an onboard IMU unit, as well as integrated depth image processing. Which covers everything Hydra needs to work. Secondly, while it is possible to gather odometry data from elsewhere, Hydra is well integrated with Kimera-VIO which has adequate documentation and instructions about running it with the Realsense D435i [2].

Hydra is available as a package for ROS, with an accompanying collection of helper packages. In its current state Hydra is completely ROS-dependent. Meaning that to run Hydra, a ROS environment must first be set up.

The Desktop-Full install of ROS Noetic as well as all optional packages were installed from the official ROS website, through the available debian package according to the available instructions [3].

[2] https://github.com/MIT-SPARK/Kimera-VIO-ROS/blob/master/docs/hardware_setup.md
[3] http://wiki.ros.org/noetic/Installation

3.3. Setting up Realsense drivers and ROS wrapper

To access the full potential of the Realsense camera, the Intel Realsense SDK (librealsense2) was installed [4]. It is possible to install it from a debian package by using APT:

```
sudo apt-get install ros-$ROS_DISTRO-realsense2-camera
```

However, this version will likely be outdated, and during the experimental part of this thesis it turned out that communications with the camera over USB did not work properly later down the line when using this version.

Librealsense2 was instead installed from scratch by downloading the source code, compiling it and installing it. The detailed instructions including requirements and shell commands are available at the librealsense Github repository [5]. The instructions specific to Ubuntu 20.04 were followed. Figure 14 shows a flowchart of the general steps taken.

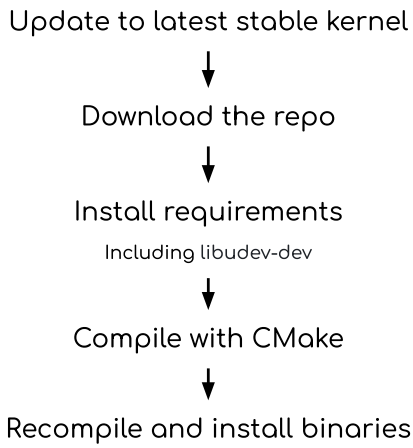

Figure 14. *Flowchart on installing librealsense2*

For easy ROS integration the realsense ROS wrapper was installed as well. The detailed instructions are available at the realsense-ROS wrapper Github repository under “Step 2: Install Intel® RealSense™ ROS from Sources” [6]. A dedicated catkin workspace was made for the Realsense ros wrapper.

[4] https://github.com/IntelRealSense/librealsense
[5] https://github.com/IntelRealSense/librealsense/blob/master/doc/installation.md
[6] realsense-ros-wrapper

Ddynamic_reconfigure was also installed using APT since there was a degree of uncertainty whether it was installed prior:

```
apt-get update apt-get install ros-noetic-ddynamic-reconfigure
```

Provided in figure 15 is a flowchart of the general steps involved.

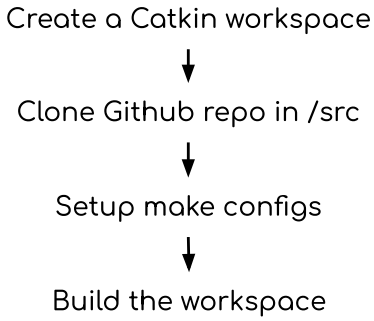

Figure 15. *Flowchart on installing realsense ROS wrapper*

3.3.1. Realsense Ros Wrapper launch file

The launch file located at *realsense-ros/realsense2_camera/launch/rs_aligned_depth.launch* was used as a template for the final launch file since it turned out that there was not a lot that needed to be changed. Sticking with the original structure of the launch file, the following arguments were added or changed as described in table 1. The new launch file was titled *rs_aligned_test.launch.*

Table 1. *Changes to rs_aligned_depth.launch*

Argument	Old value	New value	Functionality
accel_fps	250	100	Accelerometer updating rate
enable_sync	true	false (can be left as is)	Syncing timestamps
enable_fisheye	true	false	Support for fisheye lenses
unite_imu_method	Non-existent	linear_interpolation	Publishing Imu messages on the topic /camera/imu

Additionally, under the lines which specify what arguments to pass on to the *nodelet.launch.xml* file:

```
<group ns="$(arg camera)">
  <include file="$(find realsense2_camera)/launch/includes/nodelet.launch.xml">
```

It was also required to add the following line:

```
<arg name="unite_imu_method"         value="$(arg unite_imu_method)"/>
```

The Realsense ROS Wrapper now published the two infrared picture streams over /camera/infra1-2/image_rect_raw, IMU data over /camera/imu, camera info over /camera/color/camera_info, RGB color image over /camera/color/image_raw and depth image over /camera/aligned_depth_to_color/image_raw.

3.4. Setting up semantic segmentation

Semantic segmentation and everything related to it is the part of Hydra that is not publicly available. Some creative solutions therefore had to be applied. The problem faced can be broken down into a few parts:

1. A neural network must be chosen. There are many off-the-shelf networks available, so it becomes a question of finding a network that has been trained on the sorts of environments that Hydra is to be run on. It is also feasible to train a network from scratch, but this requires a lot of work.
2. A program must be designed that implements the model chosen and runs inference and labels the images acquired from the Realsense.
3. The program must be compatible with ROS.

As for the the implementation chosen in this thesis:

1. HRnet and checkpoint files trained on the MIT ADE20k dataset were used. The checkpoint files were up until recently publicly available at [7].
2. Since this thesis partly aimed at not reinventing the wheel, premade solutions that could be appropriated were seeked after. A pytorch implementation exists for semantic parsing on the MIT ADE20k dataset [8], and a forked ROS implementation has been published by *pranay731* on Github [9].
3. Ultimately, *Pranay731*'s implementation was used.

Pranay731's ROS package was set up in a new catkin workspace. The steps taken are briefly illustrated in figure 16 but are available in more detail on pranay731's Github repo.

[7] http://sceneparsing.csail.mit.edu/model/pytorch/
[8] https://github.com/CSAILVision/semantic-segmentation-pytorch
[9] https://github.com/pranay731/ros-semantic-segmentation-pytorch

Figure 16. *Flowchart on setting up semantic segmentation*

Nvidia Cuda Toolkit may be a hard requirement as well for using *Pranay731*'s implementation. It can be installed at [10].

At this point, the semantic segmentation solution accepted RGB images on the topic /camera/color/image_raw and outputted a semantically colored image on the topic /segmentation/color/image_raw. For configuring the semantics with Hydra, the authors of Hydra wrote a separate package that condenses the outputs of the semantic segmentation to only 20 labels since Hydra is limited to 20 labels, this package was not publicly available. The current solution predicted 150 labels, but this was circumvented by manually labeling the outputs later on: *see section 3.5.2*.

3.4.1. Launch file and configurations

The checkpoint files *decoder_epoch_30.pth* and *encoder_epoch_30.pth* were downloaded [11], and placed under the path *.../ros-semantic-segmentation-pytorch/ckpt/ade20k-hrnetv2-c1*.

The changes made in the launch file *semantic_segmentation.launch* are presented in table 2:

[10] https://developer.nvidia.com/cuda-downloads
[11] http://sceneparsing.csail.mit.edu/model/pytorch/

Table 2. *Changes to semantic_segmentation.launch*

Param	Old value	New value
cfg_filepath	…/ade20k-resnet50dilated-ppm_deepsup.yaml	…/ade20k-hrnetv3.yaml
model_ckpt_dir	…/ckpt/ade20k-resnet50dilated-ppm_deepsup	…/ckpt/ade20k-hrnetv2-c1

3.5. Setting up Hydra and Kimera-VIO

The code base and setup instructions for Hydra are publicly available at the Hydra Github repository [12]. The general workflow used for setting up Hydra and Kimera-VIO is explained in figure 17.

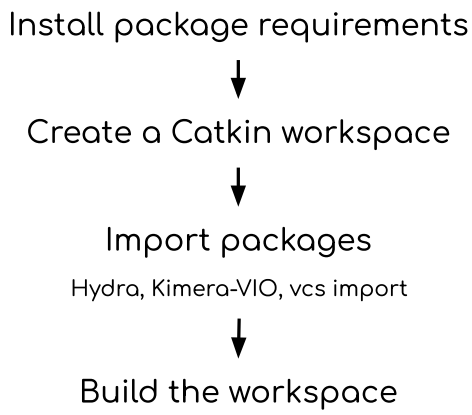

Figure 17. *Flowchart of the general steps taken when installing Hydra and Kimera-VIO*

3.5.1. Launch file and configurations for Kimera-VIO

Launching the Kimera-VIO pipeline required a ROS launch file to be configured. As Kimera-VIO comes with support for the Realsense D435i, the launch file located at *kimera-vio-ros/launch/kimera_vio_ros_realsense_IR.launch* sufficed as a starting point. The

[12] https://github.com/MIT-SPARK/Hydra

changes made to the *kimera_vio_ros_realsense_IR.launch* file are documented in table 3, and are present in the *kimera_vio_ros_realsense_IR_test.launch* file:

Table 3. *Changes made in kimera_vio_ros_realsense_IR.launch file*

Argument	Old value	New value	Functionality
viz_type	Non-existent (located in kimera_vio_ros. launch)	1	Kimera visualization, default value 0 (2d3d_mesh) is broken when running Kimera-VIO multithreaded
use_lcd	Non-existent (located in kimera_vio_ros. launch)	true	Enable loop closure detection
lcd_no_detection	Non-existent (located in kimera_vio_ros. launch)	true	Needed for Hydra loop closure detection
online	true	true/false	Automatic parsing of Rosbags. Was usually left as true except when specifically testing Rosbag parsing with Kimera-VIO
should_use_sim_time	false	true/false	Needs to be set to true if parsing a Rosbag with the --clock flag

Additionally, some static transform publishers were added to the Kimera-VIO launch file, showcased in table 4.

Table 4. *Static transform publishers added to the Kimera-VIO launch file*

Name	From <frame> to <frame>
camera_link_broadcaster	velo_link camera_link
map_link_broadcaster	world map
optical_frame_link_broadcaster	velo_link camera_color_optical_frame

3.5.2. Launch file and configurations for Hydra

The default launch file for the Hydra pipeline is located at *hydra/hydra_dsg_builder/launch/uhumans2_incremental_dsg.launch*. This launch file is very specific to uHumans2, but Nathan Hughes has provided a generic launch file for Hydra [13] in one of the issue threads on Hydra's Github [14]. The changes listed in Table 5 are based on that launch file. The custom launch file was named *hydra_realsense.launch*.

Table 5. *Changes made in hydra_realsense.launch*

Argument	Value	Functionality
sim_time_required	true/false	Like kimera-VIO's should_use_sim_time. Needs to be set to true if parsing a Rosbag with the --clock flag
sensor_frame	velo_link	Base frame with odometry
enable_dsg_lcd	true	Enabling loop closure detection and DSG optimization
start_visualizer	true	Launching RViz on launch
depth_topic	/camera/aligned_depth_to_color/image_raw	Pointing Hydra to a topic with depth images
semantic_info_topic	/camera/color/camera_info	Pointing Hydra to a topic with camera info
semantic_topic	/segmentation/color/image_raw	Pointing Hydra to a topic with a semantically segmented image
semantic_map_path	$(find kimera_semantics_ros)/cfg/ade150_colour_mapping.csv	Pointing Hydra to the file containing information to map every color in the semantically segmented image to one of 20 labels
typology_dir	$(find hydra_dsg_builder)/config/realsense_test	Pointing Hydra to the directory containing Hydra topology configurations, including label semantics
typology_config	realsense_typology.yaml	Same as above

[13] https://gist.github.com/nathanhhughes/6656b77bab2d4cc734c123f550985cb4
[14] https://github.com/MIT-SPARK/Hydra/issues/1

Argument	Value	Functionality
dsg_output_dir	$(find hydra_dsg_builder)/output/uhumans1	Directory where the dsg mesh is saved
dsg_output_prefix	SOME_PREFIX	Prefix of output
rviz_dir	$(find hydra_dsg_builder)/rviz	Pointing to a rviz configuration file
rviz_file	realsense.rviz	Same as above
config_dir (under arguments passed to dsg_builder.launch)	$(find hydra_dsg_builder)/config/realsense_test	DSG config

Additional notes about launch arguments:

Hydra ran without semantics as well, by simply pointing the semantic_topic to a topic with unprocessed RGB images. Of course this meant that there were no colors in the semantic mesh, and also no object detection.

“semantic_map_path” should point to a csv file containing color codings for semantic segmentation labels and their respective ids for Hydra. The semantic segmentation solution used resulted in 150 labels that had to be condensed. Due to time constraints not all labels were condensed, but a few were in order to enable Hydra to build a colorful mesh, and allow visualization of objects recognized as objects by Hydra. The semantic map files are initially located in the Kimera Semantics package under kimera_semantics/*kimera_semantics_ros/cfg*, and Kimera Semantics is actually responsible for annotating the metric-semantic mesh.. In table 6 the first 7 rows of *ade150_colour_mapping.csv* are showcased.

Table 6. *The first 7 rows of ade150_colour_mapping.csv.*

name	red	green	blue	alpha	id
wall	120	120	120	255	1
building	180	120	120	255	2
sky	6	230	230	255	3
floor	80	50	50	255	4
tree	4	200	3	255	5
ceiling	120	120	80	255	6

The “typology_dir” and “typology_config” arguments should point to a yaml file containing descriptors and mappings of semantic labels. In the case of the uHumans2 launch file it points to uhumans2_<location>_topology.yaml. In this file it is possible to configure structures, objects, stuff, walls and floors for Hydra. Showcased in figure 18 is the *realsense_typology.yaml* file which was set up for this project. These labels were made to mostly correspond to what information is available in the semantic map file showcased in table 6.

Figure 18. *Label descriptions in realsense_typology.yaml.*

```
dynamic_semantic_labels: [20]
structure_labels: [0, 2, 3, 8, 9, 12, 15, 17, 19]
object_labels: [13, 16, 18]
stuff_labels: [0, 1, 2, 3, 6, 8, 9, 12, 15, 17, 19]
walls_labels: [1]
floor_labels: [4]
```

3.6. Evaluation

Hydra was evaluated by recording the topics /camera/infra1/image_rect_raw, /camera/infra2/image_rect_raw, /camera/imu, /camera/color/camera_info, /camera/color/image_raw and /camera/aligned_depth_to_color/image_raw to Rosbags and then replaying them and running semantic segmentation, Kimera-VIO and Hydra on the testing machine.

For room-scale applications the meshing corresponds well with reality. In figure 19, the mesh generated by Hydra is compared to a picture. In the mesh the floor is visualized as brown, doors are green and walls are gray.

Figure 19. *To the left, Hydra's mesh. To the right, a picture of the same location*

The large-scale datasets were collected at the Delta Center in Tartu. No loop closure events were triggered in any of the datasets collected; hence, most of the resulting Scene Graphs suffer from some sort of inconsistency. Usually this manifested in some drift upwards over time. This led to meshes being built on top of each other and Hydra identifying the same locations as different rooms. This is illustrated in Figure 21. In figure 20 a picture of a scene graph is presented assuming multiple rooms in the same room. In figure 22 the semantic mesh is overlaid on the Delta Center floor plan.

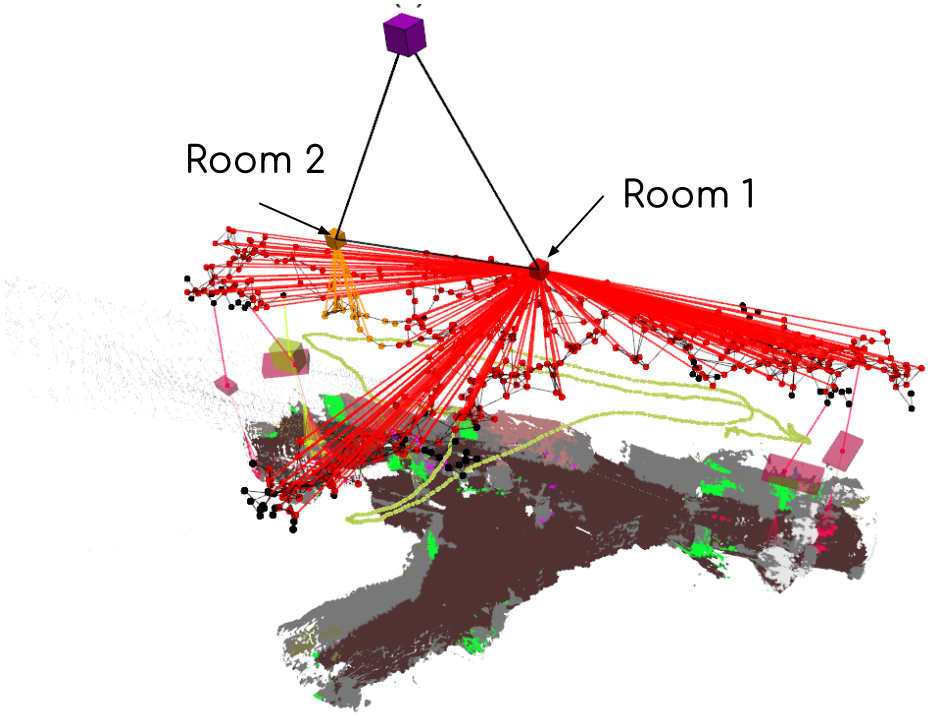

Figure 20. *Hydra scene graph, false identification of the same room as two rooms*

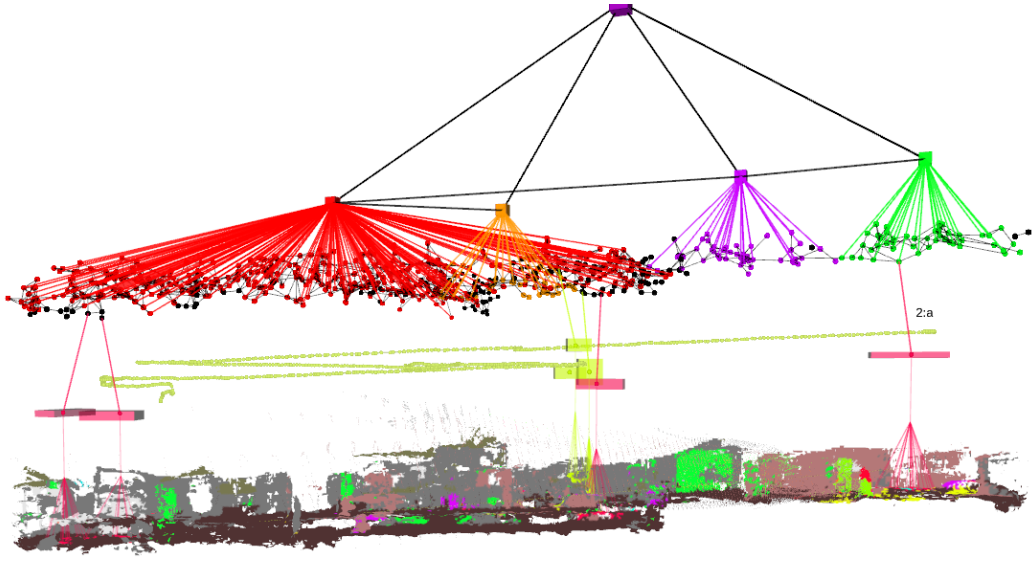

Figure 21. *Hydra scene graph and vertical drift made visible by the yellow odometry trail*

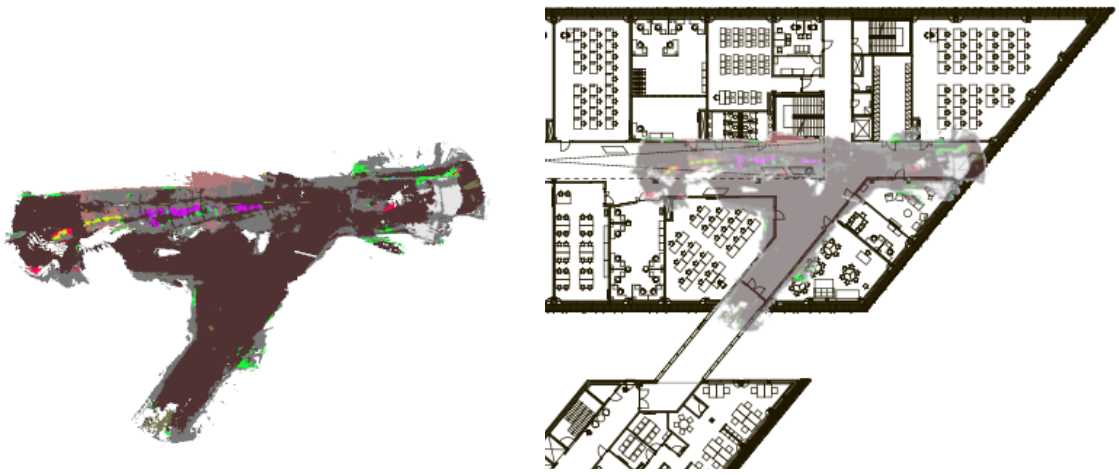

Figure 22. *Semantic mesh overlaid on the Delta Center floor plan* [30]

3.6.1. Published changes

All changes and configurations made that are referenced to in section 3. to Hydra, and accompanying packages are publicly available at a forked version of Hydra https://github.com/carlhjal/Hydra. The semantic segmentation package is not included by default.

4. Discussion and conclusion

Initially the goals of this thesis were set rather ambitiously with a goal of trying to apply Hydra to a robot and implement pathfinding or something in the likes of. Although this is not a long stretch from the current state of the at least to some extent properly configured Hydra, due to the nature of Hydra only being partly open source. A lot of the documentation that is needed to set up Hydra within a sensible time frame is not available right now.

Most parameters and configurations had to be figured out by changing one line at a time and observing any changes. That is why documenting and establishing the workflow became one of the goals of the thesis. And in this regard the thesis was fruitful.

4.1. Takeaways on Hydra and Kimera-VIO

The first takeaway is that running Hydra with Kimera-VIO and semantic segmentation uses a lot of processing power. After 3-4 minutes the Hydra backend slows down to a grind. In its current state I am not quite sure whether Hydra has much utility apart from research. Performance can most likely be improved by optimizing certain configurations. It remains unconcluded why no no loop closures were observed, again it could be due to bad configurations.

Kimera-VIO works well overall and minimal drift was observed. The Realsense D435i has an IR transmitter for depth reconstruction. Unfortunately, the transmitted IR beams are registered by Kimera-VIO as static landmarks, because of that they had to be disabled. This led to a slight detriment in the depth image. Another observation on Kimera-VIO is that if it cannot identify any landmarks for even a short period of time the odometry estimation will drift off and get ruined. This can occur for example if observing a featureless wall.

4.2. Suggestions for future work

A package for condensing and recoloring the semantically segmented images is part of Hydra but is not publicly available. Work was started on making such a package but it was not finished.

The original goal remains as a suggestion for further work. One application made possible by the scene graph would be to implement pathfinding with a high-level interface.

REFERENCES

[1] ‘Google Trends’, *Google Trends*. https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0mkz&hl=en (accessed May 23, 2023).

[2] R. Velik, ‘Towards Human-like Machine Perception 2.0’, *Int. Rev. Comput. Softw.*, vol. 1, pp. 13–21, Jul. 2010.

[3] M. Ben-Ari and F. Mondada, ‘Robots and Their Applications’, in *Elements of Robotics*, M. Ben-Ari and F. Mondada, Eds., Cham: Springer International Publishing, 2018, pp. 1–20. doi: 10.1007/978-3-319-62533-1_1.

[4] ‘These 5 robots could soon become part of our everyday lives’, *World Economic Forum*, Feb. 09, 2022. https://www.weforum.org/agenda/2022/02/robots-future-tech/ (accessed Mar. 05, 2023).

[5] N. Hughes, Y. Chang, and L. Carlone, ‘Hydra: A Real-time Spatial Perception System for 3D Scene Graph Construction and Optimization’, in *Robotics: Science and Systems XVIII*, Robotics: Science and Systems Foundation, Jun. 2022. doi: 10.15607/RSS.2022.XVIII.050.

[6] S. Huang and G. Dissanayake, ‘Robot Localization: An Introduction’, in *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, Ltd, 2016, pp. 1–10. doi: 10.1002/047134608X.W8318.

[7] ‘How the GPS assisted navigation works’. https://www.husqvarna.com/ie/learn-and-discover/how-the-gps-assisted-navigation-works/ (accessed Mar. 25, 2023).

[8] ‘GPS.gov: GPS Accuracy’. https://www.gps.gov/systems/gps/performance/accuracy/#sa (accessed Mar. 25, 2023).

[9] F. Morgado, P. Martins, and F. Caldeira, ‘Beacons positioning detection, a novel approach’, *Procedia Comput. Sci.*, vol. 151, pp. 23–30, Jan. 2019, doi: 10.1016/j.procs.2019.04.007.

[10] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter, ‘Jointly Optimizing Placement and Inference for Beacon-based Localization’. arXiv, Sep. 20, 2017. Accessed: May 24, 2023. [Online]. Available: http://arxiv.org/abs/1703.08612

[11] N. Rajagopal, ‘Localization, Beacon Placement and Mapping for Range-Based Indoor Localization Systems’, thesis, Carnegie Mellon University, 2019. doi: 10.1184/R1/9735845.v1.

[12] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer Science & Business Media, 2008.

[13] D. M. Helmick, Y. Cheng, D. S. Clouse, L. H. Matthies, and S. I. Roumeliotis, 'Path following using visual odometry for a Mars rover in high-slip environments', in *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*, Mar. 2004, pp. 772-789 Vol.2. doi: 10.1109/AERO.2004.1367679.

[14] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, 'An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics', *Intell. Ind. Syst.*, vol. 1, no. 4, pp. 289–311, Dec. 2015, doi: 10.1007/s40903-015-0032-7.

[15] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, 'Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping'. arXiv, Mar. 03, 2020. Accessed: Mar. 25, 2023. [Online]. Available: http://arxiv.org/abs/1910.02490

[16] R. Barber *et al.*, *Mobile Robot Navigation in Indoor Environments: Geometric, Topological, and Semantic Navigation*. IntechOpen, 2018. doi: 10.5772/intechopen.79842.

[17] B. Kuipers and Y.-T. Byun, 'A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations', *Robot. Auton. Syst.*, vol. 8, no. 1–2, pp. 47–63, Nov. 1991, doi: 10.1016/0921-8890(91)90014-C.

[18] S. Thrun, W. Burgard, and D. Fox, 'A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping.', presented at the Proceedings - IEEE International Conference on Robotics and Automation, Jan. 2000, pp. 321–328. doi: 10.1109/ROBOT.2000.844077.

[19] 'Overview of mobile robot mapping types | Kshitij Tiwari, Ph.D.', Feb. 17, 2023. https://kshitijtiwari.com/all-resources/mobile-robots/robot-mapping/ (accessed Mar. 26, 2023).

[20] S. Thrun, 'Robotic Mapping: A Survey', 2002.

[21] Z. He, H. Sun, J. Hou, Y. Ha, and S. Schwertfeger, 'Hierarchical topometric representation of 3D robotic maps', *Auton. Robots*, vol. 45, no. 5, pp. 755–771, Jun. 2021, doi: 10.1007/s10514-021-09991-8.

[22] S. Panzieri, F. Pascucci, I. Santinelli, and G. Ulivi, 'Merging topological data into kalman based slam', in *Proceedings World Automation Congress, 2004.*, Jun. 2004, pp. 57–62.

[23] N. Zimmerman, M. Sodano, E. Marks, J. Behley, and C. Stachniss, 'Long-Term Indoor Localization with Metric-Semantic Mapping using a Floor Plan Prior'. arXiv, Mar.

20, 2023. Accessed: May 24, 2023. [Online]. Available: http://arxiv.org/abs/2303.10959

[24] H. Durrant-Whyte and T. Bailey, ‘Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms’, Jun. 2006.

[25] M. Helmberger, K. Morin, B. Berner, N. Kumar, G. Cioffi, and D. Scaramuzza, ‘The Hilti SLAM Challenge Dataset’, *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 7518–7525, Jul. 2022, doi: 10.1109/LRA.2022.3183759.

[26] ‘SLAM Use Cases. Part 3. Precise Autonomous Drone Delivery without GPS – Augmented Pixels’. http://augmentedpixels.com/slam-use-cases-part-3/ (accessed Mar. 25, 2023).

[27] M. Cummins and P. Newman, ‘Appearance-only SLAM at large scale with FAB-MAP 2.0’, *Int. J. Robot. Res.*, vol. 30, no. 9, pp. 1100–1123, Aug. 2011, doi: 10.1177/0278364910385483.

[28] A. Rosinol *et al.*, ‘Kimera: from SLAM to Spatial Perception with 3D Dynamic Scene Graphs’. arXiv, Oct. 20, 2021. Accessed: Mar. 05, 2023. [Online]. Available: http://arxiv.org/abs/2101.06894

[29] I. Armeni *et al.*, ‘3D Scene Graph: A Structure for Unified Semantics, 3D Space, and Camera’, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South): IEEE, Oct. 2019, pp. 5663–5672. doi: 10.1109/ICCV.2019.00576.

[30] Eesti Arhitektuuri Preemiad, ‘TÜ Delta keskuse väliruum / Eesti arhitektuuripreemiad 2020’, *Eesti Arhitektuuripreemiad*. https://arhitektuuripreemiad.ee/objekt/tartu-ulikooli-delta-keskus/ (accessed May 23, 2023).

Appendix

Hydra_realsense.launch

```
<launch>
   <arg name="sim_time_required" default="false"/>
   <param name="use_sim_time" value="$(arg sim_time_required)"/>

   <!-- pointcloud for hydra: no need to change -->
   <arg name="pointcloud_topic" value="/semantic_pointcloud"/>

   <!-- Replace these with your actual sensor information -->
   <arg name="use_gt_frame" default="false"/>
   <arg name="sensor_frame" default="velo_link"/>
   <arg name="enable_dsg_lcd" default="true"/>
   <arg name="start_visualizer" default="true"/>
   <!--arg name="sensor_frame" default="left_cam_kimera" unless="$(arg use_gt_frame)"/-->
   <arg name="depth_topic" default="/camera/aligned_depth_to_color/image_raw"/>

   <arg name="semantic_config" value="ade150_semantic_config.yaml"/>
   <!-- These should point to a 2D semantic segmentation image -->
   <arg name="semantic_info_topic" default="/camera/color/camera_info"/>
   <arg name="semantic_topic" default="/segmentation/color/image_raw"/>

   <!-- semantic configuration -->
   <arg name="semantic_map_path" default="$(find
kimera_semantics_ros)/cfg/ade150_colour_mapping.csv"/>/>
   <arg name="typology_dir" default="$(find hydra_dsg_builder)/config/realsense_test"/>
   <arg name="typology_config" default="realsense_typology.yaml"/>

   <!-- see uhumans2 launch file for how these are used -->
   <arg name="dsg_output_dir" default="$(find hydra_dsg_builder)/output/uhumans1"/>
   <arg name="dsg_output_prefix" default="SOME_PREFIX"/>

   <!-- good starter rviz file, though topics could be edited for images -->
   <arg name="rviz_dir" default="$(find hydra_dsg_builder)/rviz"/>
   <arg name="rviz_file" default="d455.rviz"/>

   <!-- turns rgb (or 2D semantics) + depth into pointcloud -->
   <include file="$(find hydra_utils)/launch/includes/rgbd_to_pointcloud.xml">
       <arg name="rgb_info_topic" value="$(arg semantic_info_topic)"/>
       <arg name="rgb_topic" value="$(arg semantic_topic)"/>
       <arg name="depth_topic" value="$(arg depth_topic)"/>
       <arg name="pointcloud_topic" value="$(arg pointcloud_topic)"/>
   </include>

   <!-- performs reconstruction and extracts places -->
   <include file="$(find hydra_topology)/launch/hydra_topology_testing.launch" pass_all_args="true">
       <arg name="semantic_color_path" value="$(arg semantic_map_path)"/>
       <arg name="config" value="uhumans2_topology_config.yaml"/>
       <arg name="config_dir" value="$(find hydra_topology)/config"/>
       <arg name="debug" value="false"/>
   </include>

   <!-- constructs rest of scene graph from reconstruction and places -->
   <include file="$(find hydra_dsg_builder)/launch/dsg_builder.launch" pass_all_args="true">
       <arg name="robot_id" value="0"/>
       <arg name="use_gt_frame" value="true"/>
       <arg name="use_oriented_bounding_boxes" value="true"/>
       <arg name="config_dir" value="$(find hydra_dsg_builder)/config/realsense_test"/>
   </include>
</launch>
```

ade150_colour_mapping.csv

name	red	green	blue	alpha	id
wall	120	120	120	255	1
building	180	120	120	255	2
sky	6	230	230	255	3
floor	80	50	50	255	4
tree	4	200	3	255	5
ceiling	120	120	80	255	6
road	140	140	140	255	7
bed	204	5	255	255	8
windowpane	230	230	230	255	9
grass	4	250	7	255	10
cabinet	224	5	255	255	11
sidewalk	235	255	7	255	12
person	150	5	61	255	20
earth	120	120	70	255	14
door	8	255	51	255	15
table	255	6	82	255	16
stairs	255	224	0	255	17
plant	204	255	4	255	18
curtain	255	51	7	255	19
chair	204	70	3	255	13
sofa	11	102	255	255	0
armchair	8	255	214	255	13

NON-EXCLUSIVE LICENCE TO REPRODUCE THESIS AND MAKE THESIS PUBLIC

I, Carl Hjalmar Love Hult,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Using and Evaluating the Real-time Spatial Perception System Hydra in Real-world Scenarios,

supervised by Karl Kruusamäe, PhD.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Carl Hjalmar Love Hult

24/05/2023